Selección del Tamaño del Banco de Registros y de la Política de Asignación de Recursos en Procesadores SMT

Jesús Alastruey[1], Teresa Monreal[1], Francisco Cazorla[2], Víctor Viñals[1] y Mateo Valero[2,3]

[1]Dept. Informática e Ingeniería de Sistemas - I3A
Universidad de Zaragoza
{jalastru, tmonreal, victor}@unizar.es

[2] BSC, Centro de Supercomputación de Barcelona
francisco.cazorla@bsc.es

[3] Dept. d'Arquitectura de Computadors
Universitat Politècnica de Cataluña
mateo@ac.upc.edu

Resumen

Este trabajo estudia el impacto del tamaño del banco de registros físico (BRF) en el rendimiento de procesadores *Simultaneous Multithreading* (SMT). Como es bien conocido, el BRF es un componente crítico en este tipo de procesadores, ya que es compartido por los distintos threads que se ejecutan. Otra variable de diseño analizada es la política de asignación de recursos a los distintos threads. Este estudio evalúa varias métricas de rendimiento: instrucciones por ciclo (IPC) e instrucciones por segundo (IPS) para prestaciones brutas, y media armónica de las deceleraciones de cada thread (*Hmean-rIPC*) para verificar que ninguno es especialmente perjudicado en la ejecución multithread. A partir de estas métricas se describe un procedimiento para seleccionar un tamaño del BRF y una política de asignación de recursos que maximice IPS a la vez que obtenga valores cercanos al máximo de las otras dos métricas.

1. Introducción

Los procesadores SMT extienden la ejecución superescalar permitiendo el lanzamiento en el mismo ciclo de instrucciones de distintos threads. La implementación de este modelo de ejecución conlleva la compartición de varios recursos del procesador, entre ellos el banco de registros físico (BRF) [7][9]. Por este motivo, el tamaño del BRF debe ser suficientemente grande para almacenar los estados consolidados y especulativos de todos los threads. Además, en cada ciclo debe suministrar los operandos a las instrucciones de los distintos threads que se ejecutan. En resumen, se requieren BRFs con un gran número de entradas y puertos. Estos requerimientos implican BRFs con elevados consumos, área y tiempos de acceso. Este último hecho puede afectar al tiempo de ciclo del procesador, limitando su frecuencia máxima y reduciendo su rendimiento.

Existe por tanto un compromiso para el diseñador del procesador en cuanto al dimensionado del BRF. Por una parte, un BRF con muchas entradas reduce el número de detenciones en renombre por falta de registros físicos, lo que mejora el rendimiento medido en instrucciones por ciclo (IPC). Por otra parte, BRFs con menos entradas pueden permitir mayores frecuencias del procesador, lo que lleva a mayores rendimientos medidos en instrucciones por segundo (IPS).

En este trabajo se estudia el impacto del tamaño del BRF en las prestaciones de procesadores SMT, analizando con detalle el compromiso IPC vs. IPS. También se analiza la capacidad de distintas políticas para asignar recursos a los threads de una forma eficiente y equitativa. La métrica *Hmean-rIPC* [8] se utiliza para cuantificar dicha capacidad.

El resto del trabajo se estructura de la siguiente manera: la Sección 2 describe las distintas políticas de asignación de recursos analizadas. En la Sección 3 se detalla el entorno de experimentación y la metodología utilizada. La Sección 4 presenta los resultados del estudio y su análisis. En la Sección 5 presentamos las conclusiones del trabajo.

2. Trabajos relacionados

Varios recursos de los procesadores SMT, como por ejemplo, la ventana de lanzamiento o el BRF, son compartidos por los distintos threads que se ejecutan. La asignación de estos recursos se controla indirectamente a través de la política de búsqueda de instrucciones, aunque algunas propuestas también ejecutan acciones adicionales para actuar

sobre la distribución de recursos (por ejemplo, anular las instrucciones de un thread).

La política *Icount* [14] da mayor prioridad en la búsqueda de instrucciones a los threads con menos instrucciones en el *front-end*. Obtiene buenos resultados para threads con alto ILP, sin embargo, tiene dificultades con threads que sufren muchos fallos de L2 (*loads*). En estos casos, *Icount* puede asignar recursos a threads que están bloqueados por un *load* que ha fallado en L2 y que no podrán progresar durante muchos ciclos. Si dicho *load* tiene muchas instrucciones dependientes, el thread al que pertenece puede acaparar muchos recursos, lo que afectaría al rendimiento total del procesador.

Stall [13] es una mejora de *Icount* que evita los problemas causados por threads con tasas altas de fallos de cache. Cuando un thread tiene un fallo de L2 pendiente, *Stall* le impide buscar más instrucciones y así evita asignarle más recursos en una situación en la que no puede progresar. Sin embargo, cuando se detecta un fallo de L2 puede ser demasiado tarde para impedir que un thread acapare la mayor parte de los recursos disponibles. O al contrario, puede ocurrir que al detener dicho thread se le impida utilizar recursos que no son requeridos por el resto de threads y se produzca una infrautilización de los mismos.

Flush [13] es una extensión de *Stall* que anula las instrucciones de un thread que ha fallado en L2 para así liberar los recursos que tenía asignados. Sin embargo, sigue siendo posible que dicho thread sea penalizado sin razón ya que los recursos liberados puede que no sean requeridos por los otros threads. Además, hay que volver a buscar las instrucciones anuladas con el consiguiente gasto adicional de energía que ello supone.

Flush++ [6] se basa en que *Stall* funciona mejor que *Flush* para cargas de trabajo que no requieren muchos recursos (caracterizadas por tasas bajas de fallos de L2) y en que *Flush* funciona mejor que *Stall* para cargas de trabajo que requieren muchos recursos (caracterizadas por frecuentes fallos de L2). Por tanto, *Flush++* trata de combinar lo mejor de *Flush* y *Stall* para obtener mejores prestaciones. El comportamiento de los threads en su interacción con la cache decide la elección de una u otra política.

Dcra [5] particiona dinámicamente los recursos entre los threads según su comportamiento con la memoria. A los threads con fallos frecuentes de L1D les asigna mayores particiones, permitiéndoles explotar paralelismo más allá de los *loads* que esperan datos de los niveles superiores de la jerarquía de memoria. Los threads con pocos fallos de L1D tienen garantizada una cuota de recursos ya que el resto de threads no pueden superar el límite de recursos que tienen asignado. Por tanto, *Dcra* impide la acaparación de recursos por parte de los threads detenidos. Además, *Dcra* calcula las particiones para cada thread anticipando su necesidad de recursos e incrementa la cuota de los threads que pueden usarlos más eficientemente.

Se ha evaluado el rendimiento de todas estas políticas con BRFs de distintos tamaños.

3. Experimentación

3.1. Simulador

Se ha utilizado un simulador que consta de un *front-end* dirigido por traza y de una versión mejorada del *back-end* de smtsim [15]. Un diccionario de bloques básicos que contiene todas las instrucciones estáticas permite la ejecución de instrucciones de camino incorrecto, aunque no con todo detalle ya que no dispone de las direcciones efectivas de los *loads* de camino incorrecto. Esta limitación no supone un excesivo problema de precisión ya que la profundidad del camino incorrecto es menor en un procesador SMT que en un superescalar convencional [4].

Los detalles de la microarquitectura simulada se muestran en la Tabla 1.

3.2. Cargas de trabajo

Los programas utilizados pertenecen a la *suite* SPEC2000. En concreto, se han empleado trazas de 300 millones de instrucciones seleccionadas siguiendo las ideas presentadas en [12] (partes representativas de los programas). Los *benchmarks* se han clasificado en *ilp* y *mem* según su tasa de fallos de L2 (menor y mayor que 1%, respectivamente). De este modo, se distinguen tres tipos de cargas de trabajo: *ilp* (todos los threads *ilp*), *mem* (todos los threads *mem*) y *mix* (compuesta por threads *ilp* y *mem*). Se han seleccionado cuatro combinaciones de dos *benchmarks* para cada tipo de carga de trabajo (Tabla 2). La simulación se da

Tabla 1. Parámetros del procesador simulado.

Parámetro	Valor	Parámetro	Valor
Número de threads	2	L1 I-cache	64 KB, asociatividad 4, 4 bancos, líneas de 64 bytes, latencia: 1 ciclo
Ancho de búsqueda	8 instrucciones, de máximo 2 threads distintos	L1 D-cache	64 KB, asociatividad 4, 4 bancos, líneas de 64 bytes, latencia 1 ciclo
Ancho de decodificación	8 instrucciones	L2 U-Cache	2048 KB, asociatividad 8, 8 bancos, líneas de 64 bytes, latencia: 20 ciclos
Ancho de lanzamiento	6 int + 3 fp + 4 mem	Memoria principal	Ilimitada, tiempo acceso: 300 ciclos
ROB	512 entradas, compartido	TLB	ITLB: 48 entradas, DTLB: 128 entradas latencia: 360 ciclos
Tamaño de colas de lanzamiento (IQ)	int: 80 entradas, fp: 80 entradas mem: 80 entradas	Unidades Funcionales (*latencia*)	6 int (*simple: 1, mul/div: 4*), 3 fp (*simple/cmp/mult: 4, div: 17, sqrt: 19*), 4 load/store (*2*)
Predicción saltos	gshare 16K entradas BTB: 256, asociatividad 4 RAS: 256 entradas	BRF	72-576 int / 72-576 fp (32 int / 32 fp lógicos por thread) tiempo acceso: 1 ciclo

Tabla 2. Cargas de trabajo multithread.

Tipo de carga de trabajo	Grupo 1	Grupo 2	Grupo 3	Grupo 4
ILP	gzip, bzip2	wupwise, gcc	ammp, mesa	apsi, gcc
MEM	gzip, twolf	wupwise, twolf	lucas, crafty	equake, bzip2
MIX	mcf, twolf	art, vpr	art, twolf	swim, mcf

por concluida cuando uno de los threads finaliza su ejecución.

3.3. Métricas

Hemos utilizado tres métricas distintas para evaluar el rendimiento.

- IPCtotal: suma de las instrucciones por ciclo de cada thread. Cuantifica el rendimiento bruto (*throughput*) alcanzable por una microarquitectura.
- IPStotal: incluye el efecto del tiempo de ciclo del procesador en el rendimiento bruto. Para calcularlo, se ha supuesto que el tiempo de ciclo de procesador viene determinado por el tiempo de acceso al BRF, que se ha estimado para cada tamaño de BRF a partir del modelo de Rixner para 0.18µ [11].
- *Hmean-rIPC*: media armónica de los IPCs relativos de cada thread[1]. Combina prestaciones y equidad en la asignación de recursos entre los threads [8], por lo que penaliza las políticas que consiguen altos rendimientos en IPC acelerando los threads con alto ILP a costa de frenar a los de bajo ILP.

4. Resultados

La Figura 1 muestra IPCtotal, IPStotal y *Hmean-rIPC* para las cargas de trabajo *ilp* y *mem*. No se incluyen los resultados para la carga de trabajo *mix* por falta de espacio. Pueden encontrarse en [1]. Para cada una de las métricas, se describe su comportamiento al variar el tamaño del BRF y se compara el rendimiento obtenido por cada una de las políticas de asignación de recursos.

4.1. IPC

Como era de esperar, conforme aumenta el número de entradas del BRF, el IPC mejora. Esto

[1] Cociente entre los IPCs obtenidos al ejecutarse en modo multi-thread y single-thread.

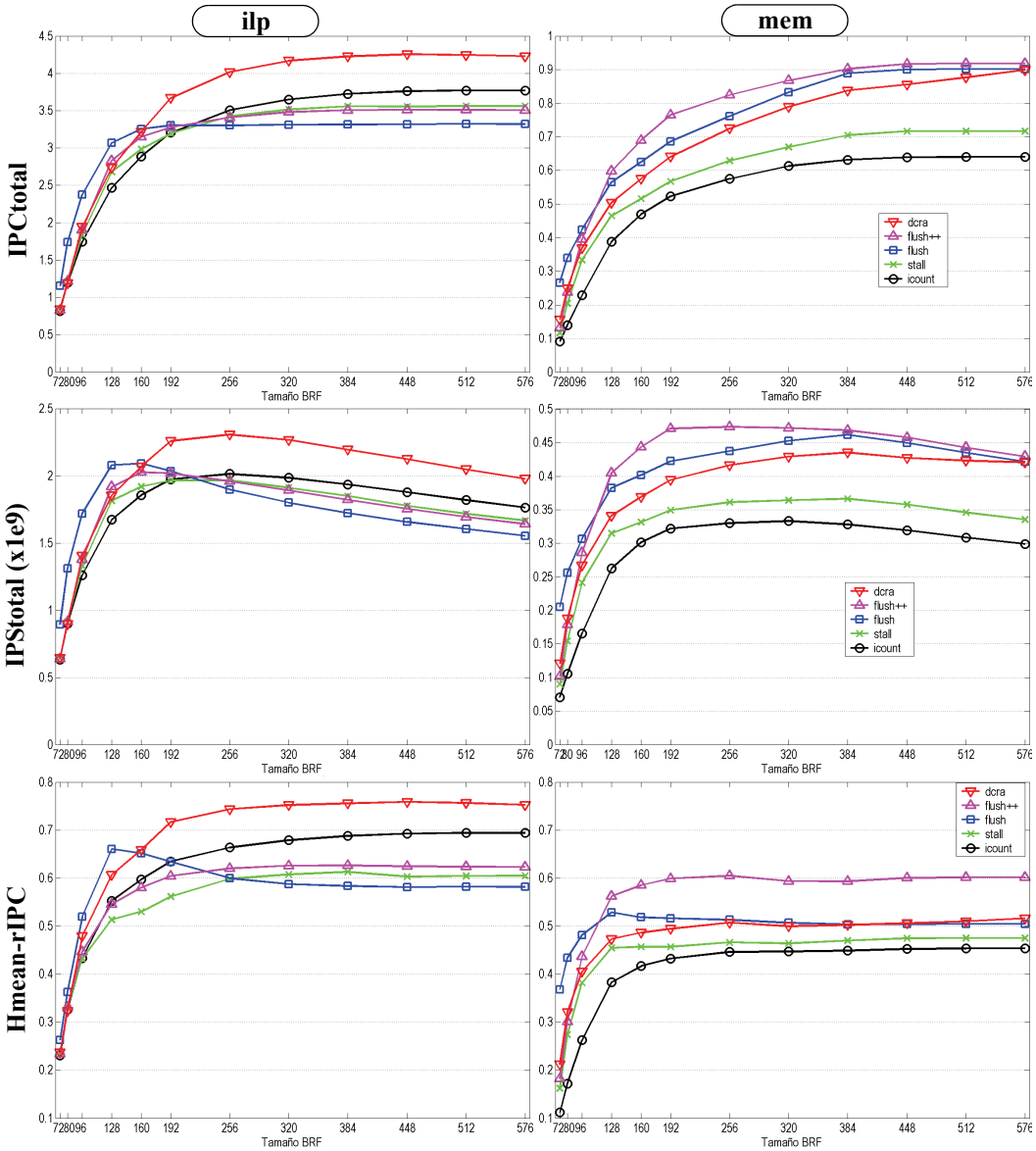

Figura 1. IPCtotal (arriba), IPStotal (centro) y Hmean-rIPC (abajo) vs. tamaño de BRF para las políticas *Icount, Stall, Flush, Flush++* y *Dcra*. Las tres gráficas de la izquierda corresponden a cargas de trabajo *ilp* y las tres de la derecha a cargas de trabajo *mem*. Nótese las escalas distintas de IPCtotal e IPStotal para *ilp* y *mem*.

se debe a que las detenciones en renombre provocadas por la escasez de registros disminuyen al haber más registros disponibles. Para BRFs ajustados (hasta 160 registros), el rendimiento se incrementa de manera considerable al aumentar el número de registros. Conforme se sigue aumentando dicho número, el rendimiento aumenta en menor escala hasta un punto en que no se obtiene ganancia de prestaciones por el incremento de registros. Este punto varía según las cargas de tra-

bajo. Por ejemplo, para *ilp*, las políticas de asignación de recursos llegan al 90% de su IPC máximo con entre 128 y 256 registros, mientras que para *mem* son necesarios entre 256 y 384 registros. Este resultado confirma que los threads de tipo *mem* necesitan más registros físicos que los de tipo *ilp* para explotar paralelismo.

Si comparamos ahora el rendimiento de las distintas políticas de asignación de recursos, tenemos que para cargas de trabajo *ilp*, con BRFs ajustados (hasta 160 registros), el mejor rendimiento se consigue con *Flush*. Para BRFs mayores, *Dcra* obtiene mucho mayor rendimiento que el resto, mientras que *Flush* se convierte en la peor alternativa. Este comportamiento de *Flush* se debe a que la anulación de instrucciones que fallan en L2 y la consiguiente liberación de sus registros asignados sólo es una acción efectiva cuando la presión sobre el BRF es alta. Para cargas de trabajo *mem*, se observa que con BRFs muy ajustados (hasta 96 registros), el mejor IPC se consigue con *Flush*. Para BRFs mayores, *Flush*++ da los mejores resultados, aunque con rendimientos similares a *Flush* y *Dcra*. El mayor número de fallos de L2 provoca que *Icount* y *Stall* se comporten mal con las cargas de trabajo *mem*.

4.2. IPS

Para calcular esta métrica se ha considerado que el acceso al BRF constituye el camino crítico del procesador y que por tanto, su tiempo de acceso determina el tiempo de ciclo del procesador. Los tiempos de acceso para cada tamaño de BRF se han calculado según el modelo de Rixner para 0.18µ [11] y pueden consultarse en [1].

En las gráficas centrales de la Figura 1 se observa que los BRFs más pequeños obtienen rendimientos bajos. Esto se debe a que, a pesar de que su tiempo de acceso permite frecuencias altas de procesador, es más acusado el efecto del elevado número de detenciones en renombre. Por otra parte, el elevado tiempo de acceso de los BRFs más grandes limita la frecuencia del procesador, lo que afecta negativamente a su rendimiento. Un punto de diseño interesante es aquel tamaño de BRF que maximiza IPS. Según nuestros resultados, este tamaño varía según el tipo de carga y la política de asignación de recursos. Para *ilp*, el máximo IPS se alcanza con entre 160 y 256 registros y para *mem* con entre 256 y 384.

La comparativa entre políticas de asignación de recursos coincide con la de la subsección anterior, ya que IPC e IPS sólo se diferencian en un factor de escala (la frecuencia del procesador).

4.3. Hmean-rIPC

Para cargas *ilp*, esta métrica vuelve a ser baja con BRFs pequeños y aumenta hasta saturar con en torno a 320 registros, excepto para *Flush*, que alcanza su máximo con 128 registros y luego disminuye hasta saturar con 320 registros. Con BRFs ajustados (hasta 160 registros), el mejor *Hmean-rIPC* se consigue con *Flush*. Para BRFs mayores, al igual que con el IPC, *Dcra* da mejores resultados y *Flush* vuelve a convertirse en la peor alternativa por el mismo motivo expuesto para el IPC.

La variación de *Hmean-rIPC* con el número de registros para las cargas de trabajo *mem* es similar a la de las cargas *ilp*, excepto la degradación de *Flush* a partir de 128 registros es menor. Con BRFs muy ajustados (hasta 96 registros), el mejor *Hmean-rIPC* se consigue con *Flush*. Para BRFs mayores, *Flush*++ obtiene casi un 20% más que *Flush* y *Dcra*. *Icount* y *Stall* vuelven a ser las peores opciones con esta carga de trabajo.

4.4. Dimensionado del BRF y selección de la política de asignación de recursos

Nuestra intención aquí es determinar el tamaño del BRF y la política de asignación de recursos que maximice las prestaciones del procesador. Esta elección la realizaremos en dos fases. En primer lugar, para cada política de asignación de registros, buscaremos un tamaño del BRF que obtenga un compromiso de prestaciones cuantificadas según las tres métricas utilizadas en este estudio. En segundo lugar, se seleccionará la política que mejor rendimiento consigue.

Si se quiere maximizar el rendimiento bruto, se debería seleccionar el tamaño del BRF con que se obtiene el máximo de prestaciones medidas con IPS. Sin embargo, en procesadores con BRFs muy pequeños es posible que sea otra estructura la que fije el tiempo de ciclo del procesador. En tal caso, la hipótesis utilizada para obtener el IPS dejaría de ser cierta. Para evitar esta posibilidad, se buscará

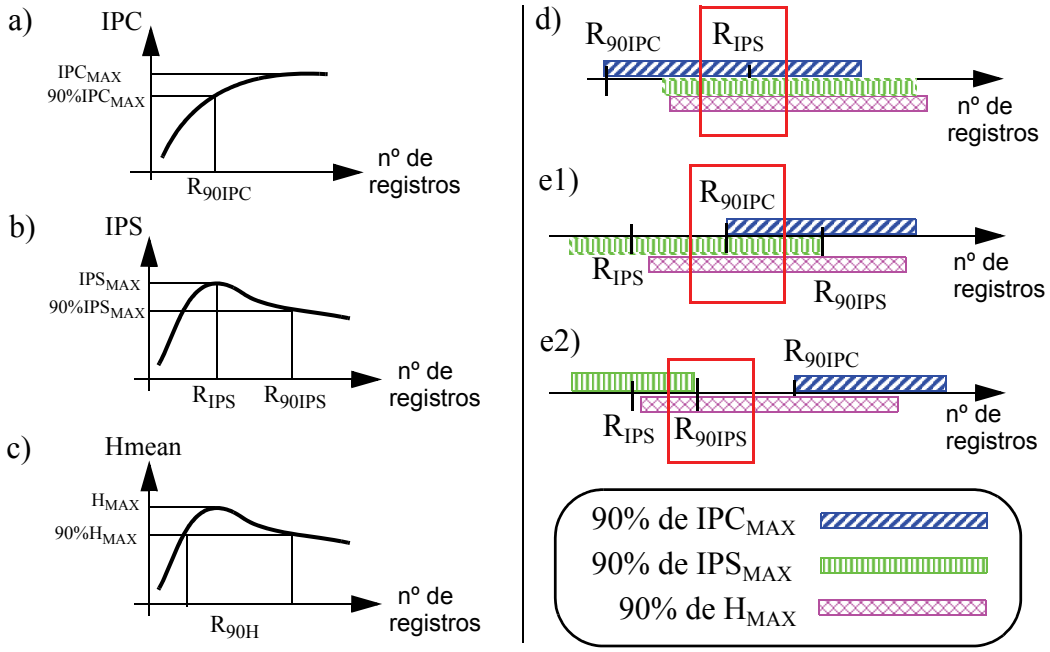

Figura 2. Elección del tamaño de banco de registros que maximiza las prestaciones multithread en base a parámetros R_{90IPC}, R_{IPS}, R_{90IPS} y R_{90H} (a, b y c). Se distinguen dos situaciones: R_{IPS} mayor o igual que R_{IPC} (d), y R_{IPS} menor que R_{IPC} (e1 y e2).

Tabla 3. Número mínimo de registros necesario para obtener el 90% del IPC máximo (R_{90IPC}), número de registros con el que se obtiene el máximo de IPS (R_{IPS}), máximo número de registros con el que se obtiene el 90% del IPS máximo (R_{90IPS}) y rango de registros con los que se obtiene el 90% de *Hmean-rIPC* máximo (R_{90H}).

	ilp			**mem**			**mix**				**todas las cargas**			
	R_{90IPC}	**R_{IPS}**	**R_{90H}**	**R_{90IPC}**	**R_{IPS}**	**R_{90H}**	**R_{90IPC}**	**R_{IPS}**	**R_{90IPS}**	**R_{90H}**	**R_{90IPC}**	**R_{IPS}**	**R_{90IPS}**	**R_{90H}**
Dcra	256	256	192+	384	384	128+	192	192	n.i.	192-448	320	320	n.i.	160+
Flush++	160	160	160+	256	256	128+	160	192	n.i.	128+	256	192	448	128+
Flush	128	160	128-256	320	384	96+	128	160	n.i.	96-256	320	320	n.i.	128+
Stall	192	192	192+	320	384	128+	256	192	448	128+	256	256	n.i.	128+
Icount	256	256	192+	256	320	160+	320	384	n.i.	320+	320	320	n.i.	256+

un compromiso entre IPS e IPC, métrica esta última independiente del tiempo de ciclo del procesador. Asimismo, para evitar desequilibrios en la ejecución de los distintos *benchmarks* de un thread, se considera una métrica adicional: *Hmean-rIPC*.

El procedimiento para la selección del tamaño del BRF se basa en unos parámetros que tratan de caracterizar las gráficas de Figura 1 y que se definen a continuación:

- R_{90IPC}: número mínimo de registros con los que se obtiene el 90% del IPC máximo (Figura 2a).
- R_{IPS}: número de registros con los que se alcanza el IPS máximo (Figura 2b).
- R_{90IPS}: número máximo de registros para los cuales se obtiene más del 90% del IPS máximo[2] (Figura 2b).

- R_{90H}: rango de registros con el cual se obtiene el 90% del valor máximo de *Hmean-rIPC* (Figura 2c).

La Tabla 3 recoge estos valores para cada carga de trabajo y para el agregado de todas las cargas. Como se ha observado que salvo alguna excepción el rango de registros con los que se obtiene el 90% del *Hmean-rIPC* máximo incluye al rango del 90% del IPS máximo, basaremos nuestro procedimiento en el resto de parámetros.

En los casos en que R_{IPS} es mayor o igual que R_{90IPC}, se cumple que con R_{IPS} registros se alcanza al menos el 90% del valor máximo de IPC (Figura 2d), por lo que R_{IPS} se convierte en un punto de diseño muy atractivo. Esto ocurre para todas las políticas de asignación de recursos con las cargas de trabajo *ilp*, *mem* y *mix* excepto *Stall*/*mix* y *Flush*++/todas las cargas (ver Tabla 3).

Si R_{IPS} es menor que R_{90IPC}, con R_{IPS} registros no se alcanza el umbral del 90% de IPC (Figura 2e1 y Figura 2e2). Aquí la elección del tamaño del BRF no es tan inmediata como en el caso anterior. De hecho, es probable que el BRF deje de ser el componente que fija el tiempo de ciclo al disminuir su tamaño por debajo de un determinado umbral y que sea otra estructura la implicada en el camino crítico. Considerando esta apreciación, tenemos una ventana de diseño limitada por ambos valores. Una opción de diseño consiste en escoger R_{90IPC}, pero existe la posibilidad de que con dicho tamaño del BRF se obtenga un rendimiento en IPS muy alejado del máximo. Para evitar esta situación vamos a considerar R_{90IPS}. Pueden surgir dos posibilidades:

- $R_{90IPC} <= R_{90IPS}$: los rangos del 90% del IPC máximo y 90% del IPS máximo se solapan. R_{90IPC} es el punto de dicho solape más cercano a R_{IPS}, por lo que es un buen punto de diseño (ver Figura 2e1). Esto ocurre en las combinaciones *Stall*/*mix* y *Flush*++/todas las cargas.
- $R_{90IPC} > R_{90IPS}$: en este caso no hay intersección entre el intervalo del 90% de IPC máximo y el del 90% de IPS máximo. Un punto de diseño de compromiso sería R_{90IPS} ya que estaría más cercano a R_{90IPC} que R_{IPS} (ver Figura 2e2). Esta situación, en principio, la menos deseable, no ha surgido en ninguno de los casos analizados.

En resumen, para cada política de asignación de recursos, se ha buscado un tamaño del BRF con el que se obtiene al menos el 90% del IPS máximo, el 90% del *Hmean-rIPC* máximo y que esté lo más cercano posible al 90% del IPC máximo. Para todas las políticas estudiadas, se ha encontrado un tamaño del BRF que cumple los dos primeros requisitos y que además está en el rango con el que se alcanza el 90% del IPC máximo. Estos tamaños están resaltados con sombreado en la Tabla 3. Finalmente, entre estos puntos de diseño y para cada tipo de carga de trabajo, se han seleccionado aquéllos con los que mayor IPS se obtiene. Están marcados con trazo grueso en la Tabla 3.

5. Conclusiones y trabajo futuro

Este trabajo evalúa el impacto de dos variables de diseño en las prestaciones de un procesador SMT: el tamaño del BRF y la política de asignación de recursos.

Para BRFs ajustados, *Flush* es la política que mejor rendimiento obtiene en todas las métricas y cargas de trabajo analizadas. Para BRFs con mayor número de entradas, nuestros resultados muestran que *Dcra* y *Flush*++ son las mejores políticas para cargas de trabajo *ilp* y *mem* respectivamente.

Si buscamos los mejores rendimientos, nuestros resultados muestran que dependen en gran medida del tipo de carga de trabajo que ejecutan. Así, para cargas con pocos fallos de L2 (*ilp*) se tiene que *Dcra* con 256 registros es la mejor elección. Para cargas con fallos frecuentes de L2 (*mem*) se ha observado que *Flush*++ con 256 registros es la mejor opción y para cargas mixtas (*mix*), *Flush* con 160 registros supone la mejor alternativa. Si agregamos todas las cargas de trabajo ejecutadas, nuestro procedimiento escoge *Flush*++ con 256 registros. Aunque estos tamaños no parecen excesivamente grandes, para obtener prestaciones altas necesitan políticas de asignación de recursos complejas, y en algunos casos, con elevado consumo de energía, por ejemplo, *Flush*. Además, cuando se ejecuten cargas de trabajo con más threads, habrá más presión sobre el BRF, y posiblemente se

[2] El valor mínimo con el que se obtiene el 90% del IPS no es de interés, ya que con él se obtiene menos IPC.

obtengan menores rendimientos incluso con BRFs con muchas más entradas. En este sentido, una línea de trabajo futuro es la mejora de la gestión del BRF extendiendo a procesadores SMT el uso de mecanismos propuestos para procesadores superescalares [2][3][10].

6. Agradecimientos

Este trabajo ha sido financiado por la Diputación General de Aragón ("Grupo Consolidado de Investigación", BOA 20/04/2005), por el Ministerio de Educación y Ciencia (TIN2004-07739-C02-01/02) y por la Red de Excelencia HiPEAC de la Unión Europea (High-Performance Embedded Architectures and Compilers, FP6-IST-004408).

Referencias

[1] J. Alastruey, T. Monreal, F. Cazorla, V. Viñals and M. Valero, "Selección del Tamaño del Banco de Registros y de la Política de Asignación de Recursos en Procesadores SMT". DIIS Technical Report R-07-02, Jun. 2007, http://webdiis.unizar.es/gaz/gaZ.papeles/RR-07-02.pdf

[2] J. Alastruey, T. Monreal, V. Viñals and M. Valero, "Microarchitectural Support for Speculative Register Renaming", *Proc. 21st IEEE International Parallel & Distributed Processing Symposium* (IPDPS), Long Beach, California USA, 26-30 March 2007.

[3] J. Alastruey, T. Monreal, V. Viñals and M. Valero, "Speculative Early Register Release". *ACM Int'l Conference on Computing Frontiers* (ICCF), May 2006, pp. 291-302.

[4] J. Burns and J-L. Gaudiot. "Exploring the SMT fetch bottleneck". *Proc. 3rd Workshop on Multithreaded Execution, Architecture, and Compilation*, 1999.

[5] F.J. Cazorla, E. Fernández, A. Ramirez and M. Valero. "Dynamically Controlled Resource Allocation in SMT Processors". *37th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO), Portland. December 2004.

[6] F. J. Cazorla, E. Fernández, A. Ramirez and M. Valero. "Improving Memory Latency aware fetch policies for SMT processors". *Proc. 5th International Symposium on High Performance Computing*, Oct. 2003.

[7] R. Kalla, B. Sinharoy, and J.M. Tendler, "IBM Power5 Chip: A Dual-Core Multithreaded Processor", *IEEE Micro*, vol. 24, no. 2, Mar.-Apr. 2004, pp. 40-47.

[8] K. Luo, J. Gummaraju, and M. Franklin. "Balancing throughput and fairness in SMT processors". *Proc. International Symposium on Performance Analysis of Systems and Software*, Nov. 2001.

[9] D.T. Marr, et al., "Hyperthreading Technology Architecture and Microarchitecture", *Intel Technology J.*, vol. 6, no. 1, Feb. 2002, http://www.intel.com/technology/itj/2002/volume06issue01/

[10] T. Monreal, V. Viñals, J. González, A. González, and M. Valero, "Late Allocation and Early Release of Physical Registers", *IEEE Transactions on Computers*, vol. 53, no. 10, Oct. 2004, pp. 1244-1259.

[11] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi and J.Owens, "Register Organization for Media Processing" *Proc. 6th Int'l Symp. High-Performance Computer Architecture* (HPCA), Jan. 2000, pp. 375-386.

[12] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. 10th Int'l Conf. Architectural Support for Programing Languages and Operating Systems* (ASPLOS), Oct. 2002, pp. 45-57.

[13] D. Tullsen and J. Brown. Handling long-latency loads in a simultaneous multithreaded processor. *Proc. 34th Annual ACM/IEEE International Symposium on Microarchitecture* (MICRO), Dec. 2001.

[14] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm. "Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor". *Proc. 23th Annual International Symposium on Computer Architecture* (ISCA), pag. 191–202, Apr. 1996.

[15] D. Tullsen, S. Eggers, and H. M. Levy. "Simultaneous multithreading: Maximizing on-chip parallelism". *Proc. 22th Annual International Symposium on Computer Architecture* (ISCA), pag. 392-403, 1995.